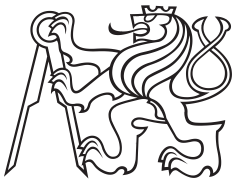
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra telekomunikační techniky

Zabezpečovací systém s logováním na SD kartu vytvořený z přípravku Nexys 4 v jazyce VHDL

Bohdan Jůza

Školitel: Ing. Pavel Lafata, Ph.D.
Květen 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jůza** Jméno: **Bohdan** Osobní číslo: **466308**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Zabezpečovací systém s logováním na SD kartu vytvořený z přípravku Nexys4 v jazyce VHDL

Název bakalářské práce anglicky:

Home Alarm with SD Card Logging Based on Nexys4 Kit and VHDL Language

Pokyny pro vypracování:

Seznamte se s kitem Digilent Nexys4 FPGA a jazykem VHDL, prostudujte možnosti ovládání přípravku a jeho periferií. Využijte kit Nexys4, následně navrhněte a realizujte domácí zabezpečovací systém s využitím různých druhů senzorů a detektorů pohybu pro detekci vniknutí do objektu, vybavený numerickou klávesnicí, akustickým alarmem a SD kartou pro logování stavu systému. Historie těchto logů půjde zpětně otevřít, eventuálně vypsat pomocí terminálového okna do konzole připojeného PC. V jazyce VHDL vytvořte potřebné knihovny a kódy pro realizaci celého systému.

Seznam doporučené literatury:

[1] Pinker, J., Poupa, M.: Číslicové systémy a jazyk VHDL. 1. vydání, Vydavatelství BEN, Praha, 2006. ISBN: 80-7300-198-5.
[2] Šťastný, J.: FPGA prakticky. 1. vydání, Vydavatelství BEN, Praha, 2010. ISBN: 978-80-7300-261-9.
[3] Král, J.: Řešené příklady ve VHDL - Hradlová pole FPGA pro začátečníky, Vydavatelství BEN, Praha, 2010. ISBN: 978-80-7300-257-2.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Lafata, Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **10.01.2019** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Pavel Lafata, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce panu Ing. Pavlu Lafatovi, Ph.D. za konzultace a věcné připomínky. Dále bych chtěl poděkovat své rodině za podporu a ČVUT v Praze za umožnění studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 24 května 2019

.....
Bohdan Jůza

Abstrakt

Bakalářská práce se věnuje problematice návrhu jednoduchého prototypu domácího zabezpečovacího systému s využitím FPGA pole na kitu Digilent Nexys 4 v jazyce VHDL. Zabývá se představením použitých komponentů a pak také návrhem samotného systému. Jsou zde vysvětleny funkcionality vytvořeného systému a podrobněji popsány stěžejní části, jako je obsluha maticové klávesnice, sedmi-segmentového displeje a logování na SD kartu.

Klíčová slova: VHDL, FPGA, Digilent Nexys 4, zabezpečovací systém, SD karta, maticová klávesnice, sedmi-segmentový displej

Školitel: Ing. Pavel Lafata, Ph.D.
Fakulta elektrotechnická,
Technická 1902/2,
16000 Praha 6

Abstract

The bachelor thesis focuses on the design of a simple prototype of a home security system using an FPGA array on a Digilent Nexys 4 kit in VHDL. It deals with the introduction of components used and with the design of the system itself. It explains the functionality of the created system and describes in more detail the main parts such as operating the matrix keyboard, seven-segment display and logging to the SD card.

Keywords: VHDL, FPGA, Digilent Nexys 4, security system, SD card, matrix keypad, seven-segment display

Title translation: Security system with logging on a SD card based on the Nexys 4 kit in VHDL language

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

V této práci jsem se zabýval návrhem a realizací prototypu domácího zabezpečovacího systému za použití kitu Digilent Nexys 4 a jazyka VHDL. Hlavním cílem bylo blíže se seznámit s daným přípravkem a možnostmi jazyka VHDL. Prostředkem pro získání nových dovedností byla tvorba zabezpečovacího systému obsahujícího různé druhy senzorů a detektorů, klávesnici, akustický alarm a SD kartu pro logování stavu systému.

Samotná práce je rozdělena do tří částí, na které následně navazuje závěrečné shrnutí a zhodnocení. První část se zabývá teoretickými základy, druhá obsahuje popis i ukázky praktické realizace a třetí pojednává o testování systému a jeho reálném použití.

Kapitola 2
Teoretický rozbor

2.1 Číslicové integrované obvody

V současné době lze pro realizaci číslicových integrovaných obvodů použít čtyři různé přístupy. První možností je složení obvodu ze standardních obvodů, například řad 74 a 4000. Druhou volbou mohou být procesory, které se více používají zhruba od přelomu 20. a 21. století. Třetí alternativou jsou obvody ASIC (*Application Specific Integrated Circuits*), které se ovšem z ekonomického hlediska vyplatí použít pouze při masové výrobě kvůli své náročnosti na vývoj a odladění. Poslední možností je použití programovatelné logiky, zejména pak obvodů PLD (Programmable Logic Devices) a FPGA (*Field Programmable Gate Arrays*). [1]

2.1.1 Programovatelná logika

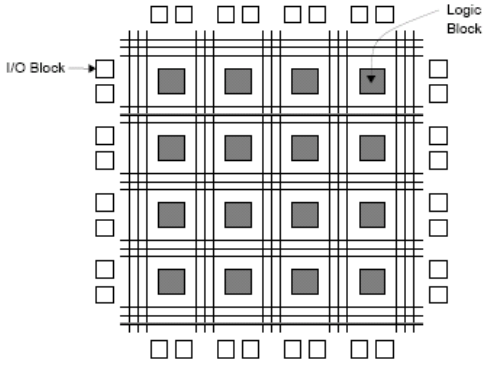
Pojmem programovatelná logika označujeme takové obvody, které jsou naprogramováním sestavitelné v číslicové obvody. V dnešní době existují dva hlavní přístupy, CPLD a FPGA. Hlavní výhodou programovatelnosti je rychlost vývoje a zejména možnost obvody ladit po celou dobu jejich vývoje, případně i během jejich nasazení v konkrétní aplikaci. [1] Pro svou práci jsem využíval kitu Digilent Nexys 4, který je vybaven hradlovým polem FPGA, jehož popisu je věnován následující oddíl.

FPGA (Field Programmable Gate Array)

FPGA, neboli česky programovatelné logické pole, je integrovaný obvod obsahující matici konfigurovatelných logických bloků (CLB) propojených pomocí programovatelných spojů. [2] Tyto bloky mohou kromě samotných buněk s logickými hradly dále obsahovat různé specializované obvody, které slouží například pro aritmetické operace. Logické funkce jsou zde tvořeny strukturou zvanou LUT (*Look-up Table*). [1] Dále zde najdeme bloky IOB (*Input/Output Block*), jejichž hlavní úlohou je udržení správné hodnoty napětí logických úrovní. Základní blokové schéma struktury FPGA je na obrázku 2.1. Pro konfiguraci hradlového pole je zde použita paměť typu RAM (bloková v rámci CLB, ale i přídavná RAM o velikosti 16 MB), jejíž obsah je po od-

pojení napájení vymazán. FPGA pole je tedy nutné po každém připojení napájecího napětí znovu naprogramovat nebo pro uchování obsahu obvodu využít přídavnou paměť Flash (*Quad-SPI Flash*).

Největším světovým výrobcem FPGA polí je firma Xilinx, jejímž čipem je osazen i přípravek Digilent Nexys 4 použitý pro vypracování této práce.



Obrázek 2.1: Základní bloková struktura FPGA obvodů [4]

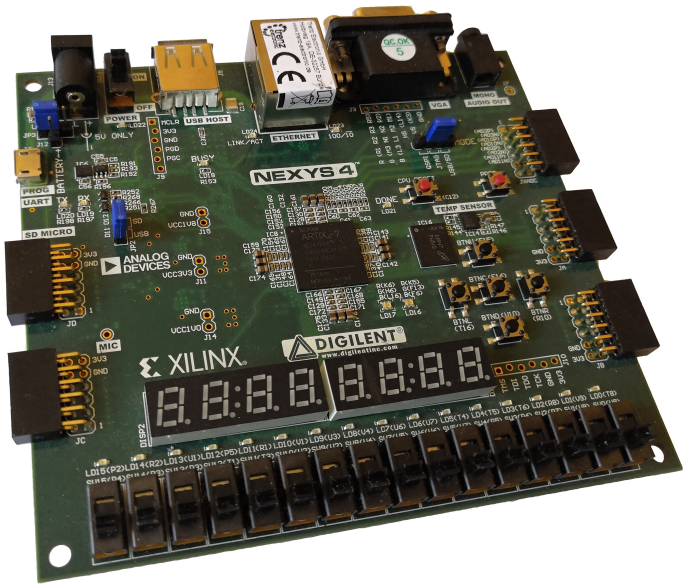
2.2 Vývojové kity

Následující sekce je věnována popisu přípravku, neboli kitu, který sloužil k realizaci mé práce. Jak je možné se dočíst například v [3], samotné hradlové pole k naprogramování logických funkcí nestačí. Pro práci s FPGA je vždy potřeba zdroj hodinového signálu, konektory pro připojení periferií a také programátor pro zavedení programu do FPGA. Kit je deska plošného spoje, která všechny tyto nezbytnosti obsahuje a často přidává i další uživatelské periferie.

2.2.1 Digilent Nexys 4

Pro svou práci jsem měl k dispozici kit od firmy Digilent s názvem Nexys 4, obrázek 2.2. V [5] je možné najít, že tato platforma je založena na integrovaném obvodu Artix-7^{TM} pocházející od firmy Xilinx. Obsahuje mnoho konektorů a periferií jako jsou:

- USB-UART převodník
- Výstup VGA
- 3-osý akcelerometr
- PWM audio výstup
- Senzor teploty
- 10/100 RJ-45



Obrázek 2.2: Kit Digilent Nexys 4

V této práci byly konkrétně použity tyto prvky:

- 2 čtyřmístné sedmi-segmentové displeje
- 2 RGB LED
- 16 LED
- 2 uživatelské spínače
- PMOD konektory pro připojení periferií

2.3 Programování FPGA

V této kapitole jsou popsány nástroje nezbytné k zavedení programu do hradlového pole z hlediska softwaru. Jedná se jednak o jazyk použitý pro popis obvodu a druhak o vývojové prostředí.

2.3.1 Jazyk VHDL

Pod zkratkou VHDL se skrývá označení *Very High Speed Integrated Circuits Hardware Description Language*. Z názvu plyne, že VHDL není programovací jazyk, ale jazyk popisný. Návrháři číslicových obvodů jeho prostřednictvím popisují chování, činnost či strukturu obvodu, která je následně syntetizována a implementována do pole FPGA. Historie jazyku VHDL sahá do 80. let 20.

století, kdy se započalo s jeho vývojem a byl také vytvořen první standard, více v [6]. Jak je možné se dočíst v [3] nebo v [6], tak kromě jazyka VHDL existuje ještě alternativní jazyk Verilog, jehož možnosti jsou rovnocenné s VHDL, avšak používá se spíše ve státech Ameriky, zatímco VHDL je typičtější pro Evropu.

Pro svou práci jsem zvolil jazyk VHDL a všechny konstrukce popsané dále se budou týkat právě tohoto jazyka.

2.3.2 Vývojové prostředí Vivado

Vivado Design Suite je vývojové prostředí od firmy Xilinx pro syntézu a analýzu podporující VHDL i Verilog. Je nástupcem programu Xilinx ISE. Obsahuje mnoho nástrojů, které vývojářům umožňují správné odladění obvodu zaváděného do FPGA nebo CPLD firmy Xilinx. Jeho omezením může být absence schématického editoru nebo to, že již nepodporuje některé starší typy čipů. [7]

2.4 Další prostředky

V této části jsou představeny prostředky použité k vypracování mé práce, které kit Nexys 4 neobsahuje.

2.4.1 Maticová klávesnice

Nedílnou součástí zabezpečovacího systému je klávesnice, jejíž pomocí je možné zadat bezpečnostní heslo a dohled systému nad monitorovaným objektem tak zapnout, či vypnout. Pojmem maticová klávesnice rozumíme množinu tlačítek uspořádaných do pravidelné struktury, v níž jsou řádky a sloupce propojeny. Díky tomuto zapojení není třeba pro každé tlačítko přivádět 2 vodiče, což vede u klávesnice o rozměrech N x N na 2 x N^2 vodičů, ale pro obsluhu stačí pouze 2 x N vodičů a algoritmus pro správné čtení stlačených tlačítek.

Použitá maticová klávesnice je na obrázku 2.3.

Obrázek 2.3: Použitá maticová klávesnice

2.4.2 Senzory

Stěžejní částí každého zabezpečovacího systému jsou senzory, kterými je monitorován zabezpečovaný objekt proti různým typům narušení. V dnešní době existuje velké množství nejrůznějších senzorů a detektorů, které jsou schopné obsáhnout veškeré požadavky na zabezpečení daného objektu a také jsou schopné poskytovat i další funkce spojené například s chytrými domácnostmi, jak je možné se dočíst v [9].

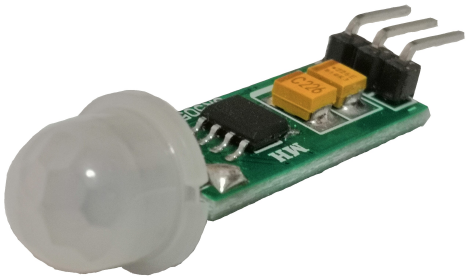
Pro účely mé bakalářské práce jsem na ukázku zvolil celkem tři zmenšené a zjednodušené senzory, jejichž profesionální varianty patří podle [10] k těm vůbec nejrozšířenějším. Jejich představení se nachází níže v této části.

Pasivní infračervené čidlo

Pasivní infračervené čidlo (PIR) je elektronický senzor, který snímá infračervené záření vyzařované objekty teplejšími, než je absolutní nula. Slovo pasivní zde vyjadřuje to, že senzor sám nevyzařuje žádné záření, ale pouze rozeznává změny v záření přicházejícím od ostatních objektů. Pohybující se objekt, v tomto případě typicky člověk, který vnikl do zabezpečeného prostoru, svým vyzařovaným IR zářením vygeneruje na senzoru puls, který je následně vyhodnocen jako narušení zabezpečeného prostoru. Na základě této události se změní stav na výstupu modulu s PIR senzorem. [11] [12]

Ve své práci jsme použil PIR senzor typu SR505, který je na obrázku 2.4. Jedná se o miniaturní typ senzoru, který je určen pro aplikace s důrazem na nízké napětí a spotřebu. Jeho hlavními parametry jsou dle [13]:

- Napájení 4,5 - 20 V
- Spotřeba $<$ 0,6 mA

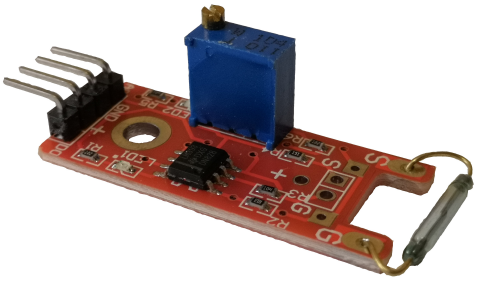


Obrázek 2.4: Modul pasivního infračerveného čidla SR505

- Úhel detekce 100°
- Vzdálenost detekce 3 m
- Zpoždění 8 s

Jazýčkový kontakt

Jazýčkový kontakt je pasivní elektrické zařízení, které se sestává ze dvou feromagnetických kontaktů uložených ve skleněné trubičce. Jejich vzájemná poloha je modulována vnějším magnetickým polem. Po přiblížení magnetického pole k jazýčkovému kontaktu dojde k zmagnetování obou jazýčků v opačné polaritě a tím tedy k jejich propojení. Po odstranění vnějšího magnetického pole se kontakty vlivem pružiny opět uvedou do výchozího rozpojeného stavu. Z výše uvedeného plyne, že pro funkci tohoto zařízení není potřeba žádné elektrické napájení a nedochází ke spotřebě energie pro udržení obou stavů. [14] Díky těmto vlastnostem je jazýčkový kontakt vhodný pro použití jako detektor otevření oken či dveří, kdy při odstranění magnetického pole z dosahu detektoru systém vyhodnotí danou událost jako narušení zabezpečeného prostoru.



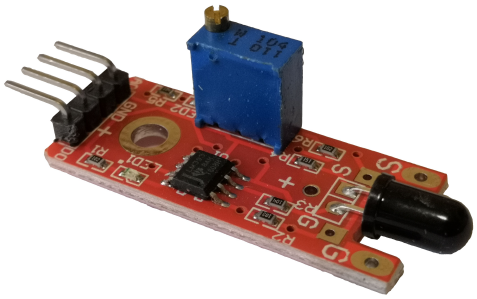
Obrázek 2.5: Modul s jazýčkovým kontaktem KY-025

Pro potřeby mé práce jsem si vybral modul KY-025 podporující 3,3 V logiku osazený jazýčkový kontaktem, který je na obrázku 2.5. Kromě toho obsahuje dále i elektroniku, díky níž je možné nastavit intenzitu přiloženého magnetického pole potřebného pro sepnutí digitálního výstupu. Dále modul

nabízí i možnost analogového výstupu, kde výstupní napětí neodpovídá žádné fyzikální veličině, spínací úroveň se poté nastavuje experimentálně. Na modulu jsou přítomny i dvě LED diody, z nichž jedna má funkci ukazatele přítomnosti napájení a druhá je propojena s hodnotou na digitálním výstupu a slouží jako indikace jeho stavu. [15]

Detektor plamene

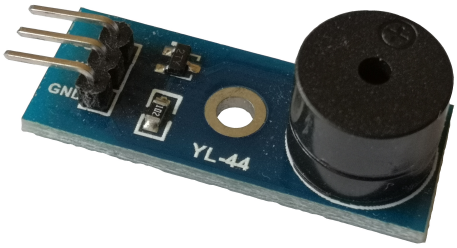
Jako třetí senzor jsem ve své práci použil detektor plamene (požární senzor). Konkrétně jsem použil modul typu KY-026, který je na obrázku 2.6. Hlavní částí tohoto modulu je 5 mm IR LED dioda, která snímá infračervené záření emitované hořením. Modul je dále osazen potenciometrem, kterým se nastavuje citlivost na přicházející záření nutné pro sepnutí tohoto modulu. Senzor podporuje analogový a digitální výstup v 3,3 V logice. [16]



Obrázek 2.6: Modul detektoru plamene KY-026

2.4.3 Aktivní piezo měnič

Piezo měniče se dělí na aktivní a pasivní. Pro tuto práci byl zvolen modul aktivního pizeo měniče YL-44, který je vidět na obrázku 2.7. Slovo aktivní znamená, že po připojení napájecího napětí a země vydává konstantně jeden tón bez potřeby externího generátoru frekvence. Kmitočet tohoto modulu se pohybuje okolo 2000 Hz. Ovládání probíhá skrze prostřední I/O pin. Přivedením logické nuly na tento pin se uvede měnič do aktivního stavu a je produkován zvuk. Naopak přivedením logické jedničky se produkce zvuku zastaví a měnič je vypnut. [19]



Obrázek 2.7: Modul Aktivního piezo měniče YL-44

2.4.4 Ovládání SD karty

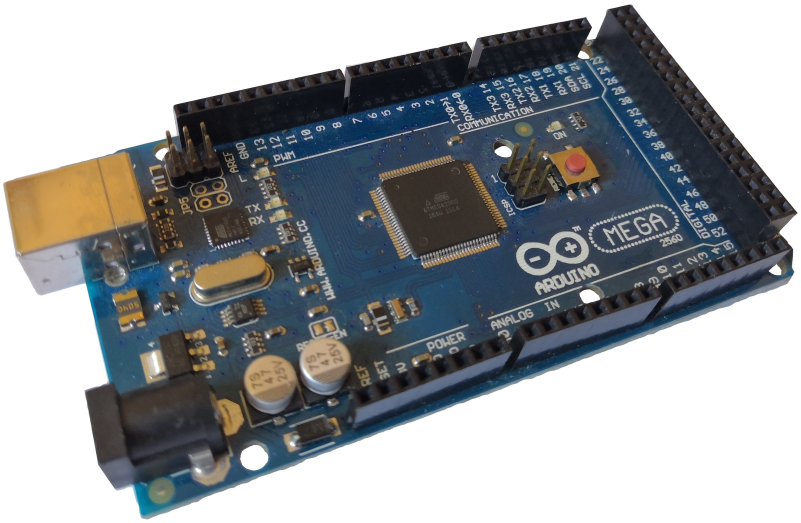
V této části budou popsány prostředky použité pro komunikaci s SD kartou, zejména pro logování informací ze zabezpečovacího systému.

Arduino Mega2560

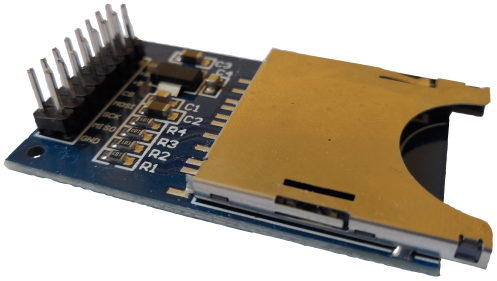
Arduino Mega2560, jehož podoba je vidět na obrázku 2.8, je vývojový kit založený na mikroprocesoru ATmega2560. Součástí tohoto mikroprocesoru je 256 kB flash paměť pro uložení kódu (z toho 8 kB je použito pro *bootloader*). Paměť SRAM (*Static Random Access Memory*) má kapacitu 8 kB a paměť EEPROM (*Electrically Erasable Read-Only Memory*) 4 kB. Kit nabízí 54 digitálních I/O pinů, z nichž 14 může být použito jako PWM výstupy, 16 analogových vstupů, 4 UART sériové porty, krystalový oscilátor na frekvenci 16 MHz, připojení pomocí USB, zdířku pro napájení a tlačítko pro reset. [20]

Modul SD karty

Pro vlastní připojení SD karty jsem vybral modul 774-012 od společnosti GM ELECTRONIC, který je na obrázku 2.9. Tento modul komunikuje přes SPI sběrnici, tedy k propojení slouží piny CS, MOSI, SCK, MISO, a podporuje 3,3 V i 5 V napájecí napětí. [18]



Obrázek 2.8: Arduino Mega2560



Obrázek 2.9: Modul čtečky SD karet GM ELECTRONIC 774-012

Kapitola 3
Praktická realizace

V této kapitole se budu věnovat vlastní realizaci zabezpečovacího systému. Bude zde vysvětlena kompletní funkcionalita systému a předvedeny ukázkové fragmenty VHDL kódu.

3.1 Přehled

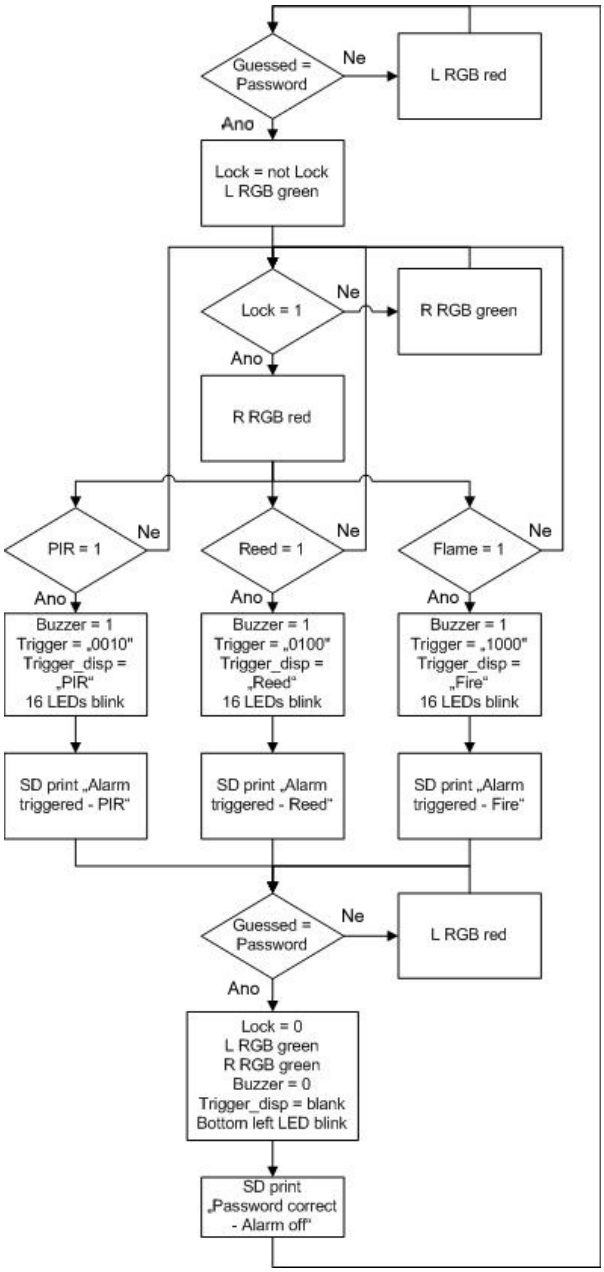
Tato podkapitola představuje přehledově celou výslednou realizaci bakalářské práce. Je zde stručně představeno chování systému z vnějšího pohledu bez znalosti vnitřních procesů. Postup, kterým bylo dosáhnuto výsledné podoby práce, je popsán v následující podkapitole 3.2.

Jádro celého zabezpečovacího systému vypracovaného v rámci této práce je popsáno v jazyce VHDL a naprogramováno na kitu Digilent Nexys 4. K tomuto kitu jsou připojeny tři různá čidla představená výše, která zajišťují samotné zabezpečení. Jedná se o PIR čidlo, které je možné typicky použit pro detekci pohybu v objektu. Dále je to jazýčkový kontakt, který by mohl plnit funkci detekce otevření dveří nebo okna. Posledním čidlem je detektor plamene, který, jak již název napovídá, najde uplatnění tam, kde je zvýšené riziko požáru.

Ve výchozím stavu je celý systém odemčen. Tuto situaci indikuje zeleně rozsvícená pravá RGB dioda na desce kitu Nexys 4. Běh programu zároveň indikuje blikající LED dioda *LD15* v levém spodním rohu přípravku. Přepínání mezi zamčeným a odemčeným stavem systému se realizuje pomocí zadání správného hesla. K zadání hesla slouží připojená maticová klávesnice. Pro zobrazení stisknutých kláves slouží pravý sedmi-segmentový displej. Zadané heslo se potvrzuje stisknutím klávesy se symbolem křížku. O správnosti zadaného a potvrzeného hesla informuje levá RGB dioda na desce kitu Nexys 4, zelená barva značí správné a červená chybné heslo. V odemčeném stavu jsou čidla inaktivní, nereagují na žádný podnět, a alarm nemůže být spuštěn. Po přepnutí do zamčeného stavu jsou čidla aktivní a pokud je zabezpečený prostor narušen, je zapnut alarm. To znamená, že se rozbliká všech 16 LED diod ve spodní části přípravku a aktivuje se zvuková signalizace z připojeného aktivního pieza. Na levém sedmi-segmentovém displeji se zobrazí název čidla,

které alarm sepnulo, respektive toho, které zaznamenalo narušení. V tuto chvíli je na SD kartu logována událost narušení s názvem čidla. Pro vypnutí alarmu je nutné zadat a potvrdit správné heslo. Tato událost je rovněž logována.

Schematické znázornění výše popsaného procesu je na obrázku 3.1.



Obrázek 3.1: Zjednodušený vývojový diagram fungování systému

3.2 Vývoj

V této podkapitole bude chronologicky nastíněn průběh vývoje práce s dělením na část v období bakalářského projektu a na část v období samotné bakalářské práce.

3.2.1 Bakalářský projekt

Nejprve bylo nutné se seznámit s kitem a hlavně s novým vývojovým prostředím Vivado, v kterém jsem si na pár pokusných projektech vyzkoušel jeho obsluhu a některé fragmenty kódů pak mohl využít jako předlohu pro tvorbu samotného zabezpečovacího systému. Poté jsem přešel k návrhu jednoduché základní kostry systému, která zahrnovala pouze aktivaci a deaktivaci alarmu. Na tento základ se postupně napojovaly další části kódu, kde dvěma hlavními body jsou maticová klávesnice a sedmi-segmentový displej. Tyto dvě části vyžadovaly hlubší teoretické znalosti a jejich implementace zabrala zbylý čas alokovaný pro etapu bakalářského projektu. Na jejím konci se tedy podařilo vytvořit základ celého systému, který již obsahoval akustický alarm, ovládání pomocí maticové klávesnice a s ní spojenou možnost zadávání bezpečnostního hesla pro změnu stavu systému a alespoň částečné využití sedmi-segmentového displeje, na kterém bylo zobrazováno právě zadávané heslo. Systém byl připraven na připojení jakéhokoli čidla schopného komunikace na principu zapnuto/vypnuto (narušení zjištěno/nezjištěno). Pro komunikaci s uživatelem byly použity dvě RGB diody, které podávaly informace o stavu systému a správnosti zadaného hesla.

3.2.2 Bakalářská práce

Na druhou část, vlastní bakalářskou práci, tedy zbývalo vyřešit zejména obsluhu SD karty. Původní záměr byl použít čtečku micro SD karet umístěnou přímo na kitu Nexys 4. Po počátečních nezdarech a zjištění komplexnosti a náročnosti ovládání SD karty tímto způsbem bylo po konzultaci s vedoucím práce rozhodnuto o použití přípravku Arduino a k němu připojené čtečce SD karet, aby byly naplněny požadavky zadání práce. S tímto řešením jsem měl větší zkušenosti a ovládání skrze Arduino není tak složité. Po vyřešení ovládní SD karty a komunikace mezi příravky došlo i na využití druhého čtyřmístného sedmi-segmentového displeje pro zobrazování názvů čidel, které byly narušeny. Čidla jsou použita celkem tři a to pasivní infračervené čidlo, jazýčkový kontakt a detektor plamene. Zbytek času byl pak věnován ladění drobných detailů a vlastnímu psaní bakalářské práce.

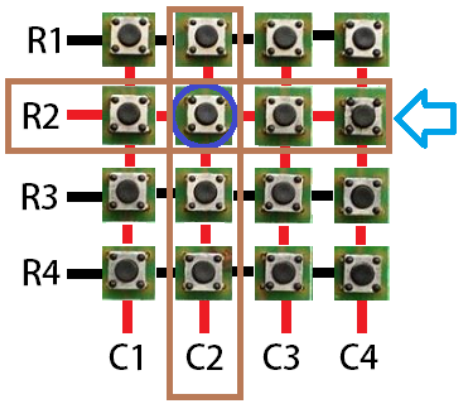
3.3 Obsluha maticové klávesnice

V této části bude detailněji popsána obsluha maticové klávesnice.

3.3.1 Čtení z maticové klávesnice

Schematické zapojení maticové klávesnice je na obrázku 3.2.

Ve své práci jsem si zvolil řádky klávesnice jakožto výstupy z přípravku a sloupce klávesnice jakožto vstupy přípravku s pull-up rezistorem. Celý princip čtení stisknuté klávesy pak spočívá v postupném přepínání řádků a čtení hodnot na jednotlivých sloupcích. Podle aktivního řádku a sloupce je pak možné identifikovat každou klávesu.



Obrázek 3.2: Schematické zapojení maticové klávesnice [8]

Pro část týkající se čtení stisku kláves jsem se rozhodl používat inverzní logiku. Princip je následující. V taktu 3,38 μs je měněna logická úroveň řádků (vektor *row_line*) a ve stejném taktu, ovšem o polovinu taktu posunutém je čtena logická úroveň na sloupcích (vektor *col_line*). Na základě toho, na jakém sloupci je pro daný řádek logická 0, je usouzeno na stisknutou klávesu.

Po vyhodnocení stisknuté klávesy je jí přiřazená hodnota zapsána do vektoru *guessed* na i-tou pozici. Hodnota i se při každém stisku inkrementuje až do čtvrté pozice (čtyřmístný displej), pak je nulována a po potvrzení hesla příslušnou klávesou se taktéž nuluje.

Eliminace zákmitů a opakovaného stisknutí je řešena vektorem *key_press*, v kterém má každá klávesa svou pozici. Ve výchozím stavu je hodnota dané pozice logická 0, je-li klávesa stisknuta, změní se hodnota na logickou 1 a po uvolnění klávesy je znovu navrácena do logické 0. Hodnota klávesy je

zapsána do vektoru *guessed* pouze při přechodu z logické 0 do logické 1, tedy pouze jednou.

Kód výše popsaného mechanismu pro klávesu D je na obrázku 3.3.

```
case temp(16 downto 14) is
    when "000" =>
        row_line <= "0111"; --first row

    when "001" =>
        if col_line = "0111" then           -- d
            if key_press(13) = '0' then
                key_press(13) <= '1';
                guessed(i) := 13;
                if i < 3 then
                    i <= i + 1;
                else
                    i <= 0;
                end if;
            end if;
```

Obrázek 3.3: Ukázka kódu pro čtení stisknuté klávesy

3.4 Sedmi-segmentový displej

V této části bude detailněji popsána obsluha sedmi-segmentového displeje.

Kit Digilent Nexys 4 nabízí dva čtyřmístné sedmi-segmentové displeje se společnou anodou. [5] Pravý displej slouží jako zobrazovač stisknutých kláves při zadávání hesla k systému. Konkrétně to znamená, že je zde zobrazován obsah podle toho, jaké hodnoty jsou v daný okamžik uloženy na příslušných místech vektoru *guessed*. Levý displej podává v případě narušení systému informace o tom, které čidlo jej sepnulo. Neboli zobrazuje obsah podle hodnot, které jsou v daný okamžik uloženy na příslušných místech vektoru *trigger_disp*.

Princip zobrazování údajů na vícemístných sedmi-segmentových displejích je následující. Nejprve je nutné si vytvořit takové časování procházení jednotlivých míst displeje, aby lidské oko nebylo schopné rozeznat jejich blikání a zobrazené údaje se jevily jako konstantně svítící. Za tímto účelem byl vytvořen signál *count_7s* jakožto vektor ve standardní logice délky 19 bitů. Jeho pomocí je vytvořen takt zhruba 763 Hz, kterým jsou jednotlivá místa displeje rozsvěcena, což je vysoko na prahem blikání lidského oka, který se běžně udává okolo 50 Hz. Rozsvěcení probíhá na základě zapisování osmibitového slova do vektoru *digit*, kde logická nula značí rozsvíceno a logická jednička zhasnuto. Každé místo displeje má přiřazenou pozici ve výše zmíněných vekto-

rech *guessed* a *trigger*. Hodnota dané pozice je podle tabulky 3.1 dekódována na osmibitové slovo, které je uloženo do vektoru *segment*. Jednotlivé bity vektoru *segment* jsou připojeny na katody sedmi-segmentových displejů a logická nula pak znamená rozsvícený a logická jednička zhasnutý segment displeje. Místa displeje jsou tedy cyklicky procházena v taktu 763 Hz a displej na nich zobrazuje znaky na základě hodnot ve vektoru *segment*.

Tabulka dekódování z vektorů *guessed* a *trigger_disp*		
Hodnota	**Konst. přiřazená vektoru *segment***	**Bitový zápis**
0	nula	11000000
1	jedna	11111001
2	dva	10100100
3	tri	10110000
4	ctyri	10011001
5	pet	10010010
6	sest	10000010
7	sedm	11111000
8	osm	10000000
9	devet	10010000
10	A	10001000
11	B	10000011
12	C	10100111
13	D	10100001
14	E	10000110
15	F	10001110
16	ii	11001111
17	P	10001100
18	r	10101111
19	star	10011100
20	hash	11001001
21	prazdny	11111111

Tabulka 3.1: Tabulka dekódování hodnot v polích *guessed* a *trigger_disp*

3.5 Arduino a SD karta

O obsluhu SD karty a logování informací ze zabezpečovacího systému se stará kit Arduino Mega2560, ke kterému je připojen modul čtečky SD karet GM ELECTRONIC 774-012.

3.5.1 Komunikace

Kity Digilent Nexys 4 a Arduino Mega2560 jsou mezi sebou propojeny dvěma vodiči, přes které dochází k jednosměrné komunikaci ve směru z kitu Nexys do Arduina. V této práci nebyl tedy použit žádný existující standardizovaný protokol, ale velmi jednoduchý typ komunikace.

Princip přenášení informací je následující. Pokud je zabezpečený prostor narušen, přepne se výstup *arduino_alarm* na kitu Nexys 4 do logické 1. Tento výstup je připojen na vstupní pin číslo 22 na kitu Arduino pojmenovaný *alarmPin*. Ve stejnou chvíli se na výstup *arduino_trigger* začne v taktu 1000 Hz odesílat čtyřbitové slovo identifikující čidlo, které bylo spuštěno. Tento výstup je připojen na vstupní pin číslo 23 na kitu Arduino pojmenovaný *triggerPin*. Poté, co Arduino zaznamená změnu stavu z logické nuly do logické jedničky na svém vstupu *alarmPin*, začne načítat sled bitů na vstupu *triggerPin*. Po přijmutí celého čtyřbitového slova identifikuje na základě skladby bitů, které čidlo zaznamenalo narušení a jeho název přiřadí do proměnné *data* podle klíče uvedeného v tabulce 3.2. Jedná se o kód typu 1 z n (v tomto případě kód 1 z 4), který byl použit z důvodu dobrého zabezpečení proti vzniku a detekci chyb při komunikaci, vyšší redundance kódu zde nevadí.

```
alarmState = digitalRead(alarmPin);

if (alarmState != lastAlarmState) {
  if (alarmState == HIGH) {
    delay(1);
    for (int i=0; i<4; i++) {
      trigger[i] = digitalRead(triggerPin);
      delay(1);
    }
```

Obrázek 3.4: Ukázka kódu pro ovládání komunikace na straně kitu Mega2560

Výše popsaný princip je vidět v následujících ukázkách kódů. Na obrázku 3.4 je fragment kódu v jazyce Wiring, který slouží pro čtení informací posílaných z kitu Nexys 4 v kitu Mega2560. Na obrázku 3.5 je znázorněn proces v jazyce VHDL, který řídí komunikaci mezi kity Nexys 4 a Mega2560 na straně kitu Nexys 4.

Tabulka identifikace čidel		
Přijaté čtyřbitové slovo	***data***	**Čidlo**
0001	Button	Servisní tlačítko
0010	PIR	Pasivní infračervené čidlo
0100	Reed	Jazýčkový kontakt
1000	Fire	Detektor plamene

Tabulka 3.2: Tabulka identifikace čidel

```
arduino_process : process(clock,arduino,trigger) is
variable count : integer := 0;
variable j : integer := 0;

begin
arduino_alarm <= arduino;
if clock='1' and clock'event then
    count := count + 1;
    if count = 100000 then
        count := 0;
        if arduino = '1' then
            if j < 4 then
                arduino_trigger <= trigger(j);
                j := j + 1;
            end if;
        else
            j := 0;
        end if;
    end if;
end if;
end process;
```

Obrázek 3.5: Ukázka kódu pro ovládání komunikace na straně kitu Nexys 4

3.5.2 Zápis na SD kartu

O zápis na SD kartu se již stará výhradně kit Arduino Mega2560. Konkrétně byla pro tyto účely použita knihovna *SD.h*, která obsahuje příkazy pro práci s SD kartou po SPI sběrnici. Arduino Mega2560 má pro použití SPI sběrnice vyhrazeny digitální piny 50 až 53. Nejdůležitější je zde pin 53, který má úlohu pinu *Chip Select*, dále CS. Pin CS, respektive číslo pinu v úloze CS, je argumentem příkazu *SD.begin(CS)* pro inicializaci komunikace mezi kitem a modulem s čtečkou SD karet. Dále je v případě požadavku na zápis nutné otevřít soubor, do kterého se bude zapisovat. To se provádí příkazem *SD.open()*, který obsahuje dva argumenty. Zaprvé je to název soubor a jeho přípona, tedy v tomto případě *LOG.txt* a zadruhé druh operace s SD kartou, kterou chceme provádět, tedy *FILE_WRITE*. Následně se do otevřeného souboru zapisují logovací informace příkazy *print()*, respektive *println()*. Po ukončení zápisu je nutné soubor uzavřít. K tomuto účelu složí příkaz *close()*.

Logovány jsou v rámci této práce v zásadě dvě informace. První je informace o tom, že byl systém narušen a které čidlo ho sepnulo. Zapisuje se hláška *„Alarm triggered – “* následovaná názvem čidla. Druhá informace má podobu *„Password correct - Alarm off“* a zpravuje o tom, že po narušení systému bylo zadáno správné heslo a alarm je vypnutý.

Kapitola 4

Praktické testování systému

Po vytvoření finální verze byl systém podroben testovacímu provozu, kdy byly zkoušeny různé situace, které mohou během používání nastat. Byly vyzkoušeny i některé nestandardní události, jako jsou například vícenásobný stisk tlačítek, narušení více čidel najednou a další. Systém obstál ve všech případech a nikdy se nestal nijak nestabilní. Ovšem i přesto se během používání vyskytly některé problémy, které by bylo vhodné v budoucnosti doladit. Jejich popis bude dále v této kapitole. V závěru této kapitoly budou rovněž ukázány různé stavy během používání systému.

4.1 Stisk kláves

První slabinou systému je ovládání pomocí klávesnice. Daný typ je určen spíše pro nenáročné použití, stisk není zcela jistý a dochází při něm k četným zákmitům. Implementované ošetření těchto přechodových jevů se ukazuje jako ne zcela dostatečné a občas dochází k nechtěnému vícenásobnému stisku klávesy.

S výše popsaným souvisí absence klávesy *smazat*, pro úpravu zadávaného hesla. V současné konfiguraci je nutné i chybné heslo potvrdit a zadat znovu správné.

Při stisku více kláves najednou dochází k vyhodnocení dle implementovaného pirncipu čtení z klávesnice. Žádně aditivní ošetření není implementováno. To znamená, že pokud jsou stisknuty 2 klávesy v jednom řádku, vypíše se první stisknutá klávesnice dvakrát, jednou při stisku a podruhé při puštění. Pokud jsou klávesy v rozdílných řádcích, vypíšou se v pořadí, v kterém byly stisknuty.

4.2 Citlivost senzorů

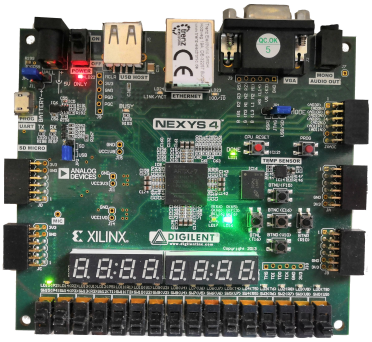
Druhým aspektem k vylepšení jsou použité senzory, které jsou stejně jako klávesnice určeny spíše pro vyvíjení prototypů a nesnesou požadavky na reálné nasazení. Toto se týká především PIR senzoru, který není nijak nastavitelný,

a navíc disponuje funkcí pamatování si stavu po určitou dobu, což někdy zapříčiní neočekáváný stav systému. Zbylé dva senzory mají nastavitelnou citlivost a jejich chování lze doladit pro potřeby konkrétního použití.

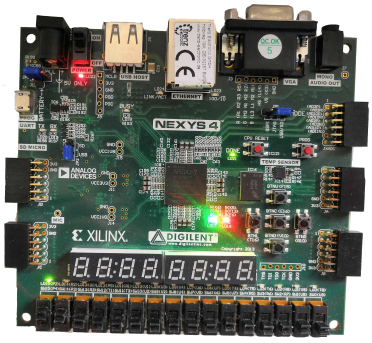
4.3 Ukázky stavů systému

Na následujících obrázcích jsou vidět různé stavy systému při jeho používání. Konkrétně na obrázku 4.1a je vidět stav systému po zapnutí a zavedení kódu. Na obrázku 4.1b je vidět stav systému po jeho uzamčení. Na obrázcích 4.2 jsou vidět stavy systému po jeho narušení, na obrázku 4.2d i se zadaným heslem systému.

Ukázka logovacich zpráv ukládaných na SD kartu je na obrázku 4.3.

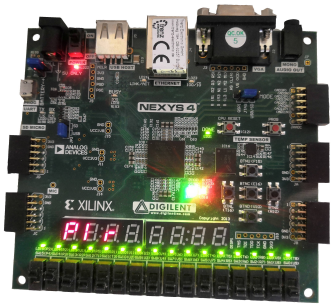


(a) : Stav systému po zapnutí

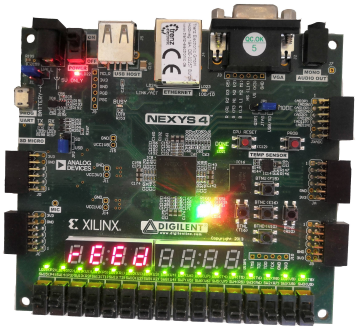


(b) : Stav systému v uzamčené podobě

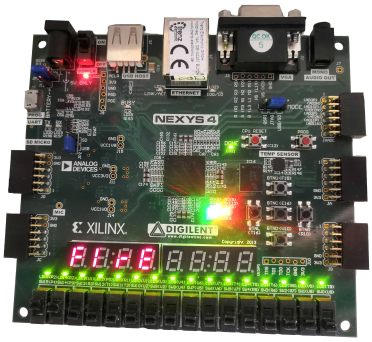
Obrázek 4.1: Systém v nenarušeném stavu



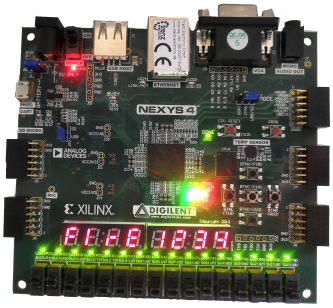
(a) : PIR



(b) : jazýčkový kontakt

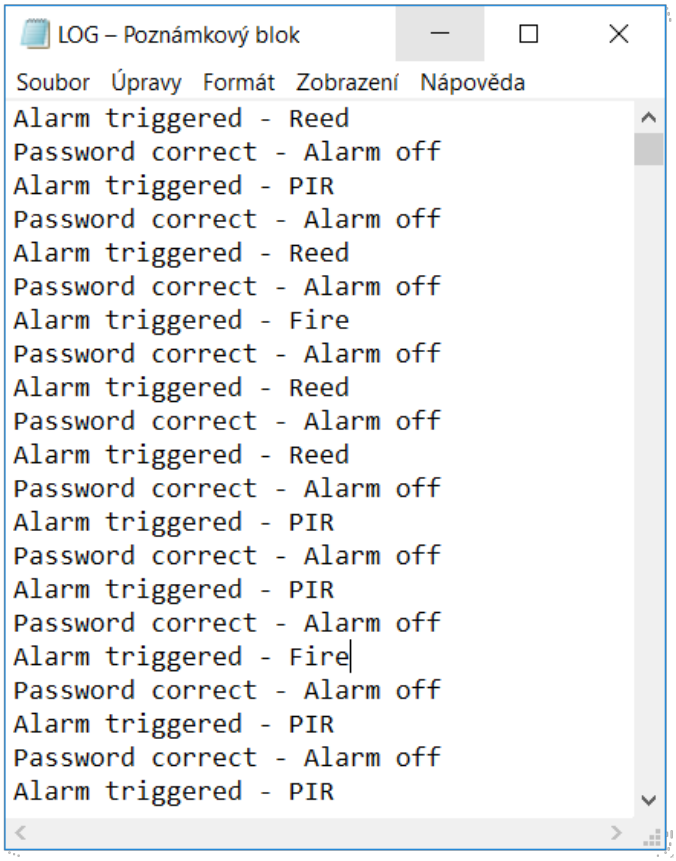


(c) : detektor plamene



(d) : detektor plamene společně se zadaným heslem

Obrázek 4.2: Stav systému po narušení senzoru



Obrázek 4.3: Ukázka logu na SD kartu

Kapitola 5

Závěr

Výsledkem práce je prototyp domácího zabezpečovacího systému, skládajícího se z řídící jednotky v podobě kitu Nexys 4 a k němu přidruženého kitu Arduino Mega2560 sloužícího pro logování na SD kartu a tří čidel pro monitorování zabezpečného prostoru. Využity byly některé další prvky obou přípravků pro komuinikaci s uživatelem. Tím došlo k naplnění cílů práce, zejména tedy k prohloubení znalostí v problematice jazyka VHDL.

Teoretická část práce se zabývá zvláště představením všech použitých prostředků, a to fyzického i programového vybavení. Praktická část se sestává z popisu systému a jeho vývoje. Pozornost je věnována stěžejním částem kódu, které jsou detailněji vysvětleny.

V průběhu realizace práce bylo třeba vyřešit několik problémů, z nichž největším byla implementace obsluhy logování na kartu SD. I z toho důvodu nezbyl již čas na další rozšíření a proto je možné tento systém v budoucnu vylepšovat i doplňovat o další funkce. Jedním z hlavnívh vylepšení by byla absence přípravku Arduino a řešení ovládání SD karty pouze v rámci kitu Nexys 4. Dále by bylo vhodné doplnit systém o modul reálného času, o jehož informace by mohly být doplněny logovací zprávy. Dalším krokem by mohla být možnost přístupu do zabezpečeného objektu na základě RFID čtečky.

Literatura

[1] KOLOUCH, Jaromír. Programovatelné logické obvody a hradlová pole – moderní stavební prvky číslicových systémů. *Automatizace: časopis pro automatizaci, měření a inženýrskou informatiku* [online]. 2009, Leden 2009, 52(1), 46 - 49. ISSN 0005125X. Dostupné také z: http://www.urel.feec.vutbr.cz/web_pages/projekty/clanky/Kolouch_PLD.pdf

[2] What is an FPGA? Field Programmable Gate Array. *Xilinx* [online]. San Jose: Xilinx, 2018 [cit. 2018-12-23]. Dostupné z: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[3] KRÁL, Jiří. *Řešené příklady ve VHDL: hradlová pole FPGA pro začátečníky.* Praha: BEN - technická literatura, 2010. ISBN 978-80-7300-257-2.

[4] Investigating Serial Microprocessors for FPGAs - Scientific Figure on ResearchGate. In: *ResearchGate* [online]. [cit. 2018-12-23]. Dostupné z: https://www.researchgate.net/figure/Basic-FPGA-structure-from-5_fig1_242531464

[5] Nexys 4^{TM} FPGA Board Reference Manual. In: *DIGILENT: A National Instruments Company* [online]. [cit. 2018-12-24]. Dostupné z: https://reference.digilentinc.com/_media/nexys:nexys4:nexys4_rm.pdf

[6] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL.* Praha: BEN - technická literatura, 2006. ISBN 80-730-0198-5.

[7] Vivado Design Suite User Guide: Getting Started. *Xilinx Inc.* [online]. 2014 [cit. 2018-12-25]. Dostupné z: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug910-vivado-getting-started.pdf

[8] Matrix-keypad-locate-keypress. In: *EMBED JOURNAL* [online]. [cit. 2018-12-29]. Dostupné z: https://embedjournal.com/assets/posts/embedded/2013-08-17-interface-4x4-matrix-keypad-with-microcontroller/matrix-keypad-locate-keypress.png

[9] CANO, L.A. Smart sensor integration into security networks. 38th Annual 2004 International Carnahan Conference on Security Technology, 2004. IEEE, 2004, 38, 82-84. DOI: 10.1109/CCST.2004.1405373. ISBN 0-7803-8506-3. Dostupné také z: http://ieeexplore.ieee.org/document/1405373/

[10] OXER, Jonathan a Hugh BLEMINGS. Security/Automation Sensors. Practical Arduino: cool projects for open source hardware. New York, NY: Distributed by Springer-Verlag, c2009, 81 - 82. Technology in Action Press book. ISBN 978-1-4302-2477-8.

[11] LUO, Ren C., Ogst CHEN a Cheng Wei LIN. Indoor human monitoring system using wireless and pyroelectric sensory fusion system. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems [online]. IEEE, 2010, 2010, , 1507-1512 [cit. 2019-03-24]. DOI: 10.1109/IROS.2010.5651345. ISBN 978-1-4244-6674-0. Dostupné z: http://ieeexplore.ieee.org/document/5651345/

[12] MICHALEC, Libor. PIR detektor: skvělý sluha, ale zlý pán. In: Vyvoj.hw.cz: profesionální elektronika [online]. 2013, 19. 3. 2013 [cit. 2019-03-24]. Dostupné z: https://vyvoj.hw.cz/automatizace/pir-cidlo-skvely-sluha-ale-zly-pan.html

[13] HC-SR505 Mini PIR Motion Sensor. In: Elecrow [online]. Shenzhen, 2018 [cit. 2019-03-24]. Dostupné z: https://www.elecrow.com/hcsr505-mini-pir-motion-sensor-p-1382.html

[14] LAI, Wei-Cheng, Chitsung HONG, Wen Ching LAI, Chun-Hao CHANG, Chun-Po CHANG, Ching-Hung CHEN a Weileun FANG. A normally closed MEMS micro reed switch with fill in liquid metal micro hinge structure. 2015 Transducers - 2015 18th International Conference on Solid-State Sensors, Actuators and Microsystems (TRANSDUCERS) [online]. IEEE, 2015, 2015, , 572-575 [cit. 2019-03-25]. DOI: 10.1109/TRANSDUCERS.2015.7180988. ISBN 978-1-4799-8955-3. Dostupné z: http://ieeexplore.ieee.org/document/7180988/

[15] KY-025 Reed module. In: SensorKit: X40 [online]. 2017 [cit. 2019-03-25]. Dostupné z: http://sensorkit.en.joy-it.net/index.php?title=KY-025_Reed_module

[16] KY-026 FLAME SENSOR MODULE. In: ArduinoModules [online]. ArduinoModules, 2018, November 16, 2018 [cit. 2019-03-31]. Dostupné z: https://arduinomodules.info/ky-026-flame-sensor-module/

[17] VODA, Zbyšek. Průvodce světem Arduina. Vydání druhé. Bučovice: Martin Stříž, 2017. ISBN 978-80-87106-93-8.

[18] Modul čtečky SD karet, SPI sběrnice. GM electronic [online]. Praha: GM electronic, 2019 [cit. 2019-04-02]. Dostupné z: https://www.gme.cz/modul-ctecky-sd-karet-spi-sbernice

[19] THAKUR, Manoj. Arduino UNO- Active Buzzer Module KY-012 YL-44. In: Steps2Make [online]. Steps2Make, 2017, September 25, 2017 [cit. 2019-04-06]. Dostupné z: https://steps2make.com/2017/09/arduino-uno-active-buzzer-module-ky-012-yl-44/

[20] ARDUINO MEGA 2560 REV3. Arduino Store [online]. Arduino, 2019 [cit. 2019-05-04]. Dostupné z: https://store.arduino.cc/mega-2560-r3

Příloha A

VHDL kód

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Security_system is
port(
    clock :             in STD_LOGIC;
    rst :               in STD_LOGIC;
    btn_on :            in STD_LOGIC;
    btn_off :           in STD_LOGIC;
    PIR :               in STD_LOGIC;
    reed :              in STD_LOGIC;
    flame :             in STD_LOGIC;
    arduino_alarm :     out STD_LOGIC;
    arduino_trigger :   out STD_LOGIC;
    buzzer :            out STD_LOGIC;
    RGB_status :        out STD_LOGIC_VECTOR(2 downto 0);
    RGB_password :      out STD_LOGIC_VECTOR(2 downto 0);
    col_line :          in  STD_LOGIC_VECTOR(3 downto 0);
    row_line :          out STD_LOGIC_VECTOR(3 downto 0);
    led :               out STD_LOGIC_VECTOR(15 downto 0);
    segment :           out STD_LOGIC_VECTOR(7 downto 0);
    digit :             out STD_LOGIC_VECTOR(7 downto 0)
);
end Security_system;

architecture Behavioral of Security_system is

type integer_vector is array(0 to 3)of integer range 0 to 21;

signal temp :       STD_LOGIC_VECTOR(29 downto 0);
signal alarm :      STD_LOGIC := '0';
signal status :     STD_LOGIC;
```

```
signal lock :       STD_LOGIC;
signal arduino :    STD_LOGIC;
signal trigger :    STD_LOGIC_VECTOR (3 downto 0);
signal count_7s:STD_LOGIC_VECTOR(19 downto 0):=(others=>'0');
signal i:integer range 0 to 4:= 0;
signal key_press:STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');

--constants for 7-seg display
constant nula :     STD_LOGIC_VECTOR(7 downto 0) :="11000000";
constant jedna :    STD_LOGIC_VECTOR(7 downto 0) :="11111001";
constant dva :      STD_LOGIC_VECTOR(7 downto 0) :="10100100";
constant tri :      STD_LOGIC_VECTOR(7 downto 0) :="10110000";
constant ctyri :    STD_LOGIC_VECTOR(7 downto 0) :="10011001";
constant pet :      STD_LOGIC_VECTOR(7 downto 0) :="10010010";
constant sest :     STD_LOGIC_VECTOR(7 downto 0) :="10000010";
constant sedm :     STD_LOGIC_VECTOR(7 downto 0) :="11111000";
constant osm :      STD_LOGIC_VECTOR(7 downto 0) :="10000000";
constant devet :    STD_LOGIC_VECTOR(7 downto 0) :="10010000";
constant A :        STD_LOGIC_VECTOR(7 downto 0) :="10001000";
constant B :        STD_LOGIC_VECTOR(7 downto 0) :="10000011";
constant C :        STD_LOGIC_VECTOR(7 downto 0) :="10100111";
constant D :        STD_LOGIC_VECTOR(7 downto 0) :="10100001";
constant E :        STD_LOGIC_VECTOR(7 downto 0) :="10000110";
constant F :        STD_LOGIC_VECTOR(7 downto 0) :="10001110";
constant ii :       STD_LOGIC_VECTOR(7 downto 0) :="11001111";
constant P :        STD_LOGIC_VECTOR(7 downto 0) :="10001100";
constant r :        STD_LOGIC_VECTOR(7 downto 0) :="10101111";
constant star :     STD_LOGIC_VECTOR(7 downto 0) :="10011100";
constant hash :     STD_LOGIC_VECTOR(7 downto 0) :="11001001";
constant prazdny :  STD_LOGIC_VECTOR(7 downto 0) :="11111111";

begin

--divider for LED blinking, turning on alarm
blink_buzzer : process (clock, status, lock)
variable count : integer := 0;
variable blink : STD_LOGIC := '0';

begin
    if clock='1' and clock'event then
        count := count + 1;

        if count = 50000000 then
            count := 0;
            blink := not blink;
        end if;
```

```
    end if;

    if status = '1' then                    -- triggered
        led <= (15 downto 0 => blink);
        buzzer <= '0';
    end if;

    if status = '0' then                    -- not triggered
        led <= (15 => blink, others => '0');
        buzzer <= '1';
    end if;
end process;

--communcation with arduino
arduino_process : process(clock,arduino,trigger) is
variable count : integer := 0;
variable j : integer := 0;

begin
    arduino_alarm <= arduino;
    if clock='1' and clock'event then
        count := count + 1;
        if count = 100000 then
            count := 0;
            if arduino = '1' then
                if j < 4 then
                    arduino_trigger <= trigger(j);
                    j := j + 1;
                end if;
            else
                j := 0;
            end if;
        end if;
    end if;
end process;

--main process containing matrix keypad and 7-seg display
main_keypad:process(clock,rst,count_7s,alarm,i,status,lock) is

--set password
constant password : integer_vector := (1,2,3,4);
--guessed password
variable guessed : integer_vector := (21,21,21,21);
--array for triggers to display
variable trigger_disp : integer_vector := (21,21,21,21);
```

```
begin

    if rising_edge(clock) then
        temp <= temp + 1; --divider for matrix keypad
        count_7s <= count_7s + 1; --divider for 7-se display

        if lock = '1' then --system locked
            RGB_status <= "100";
            --Red diode, locked, can be triggered
            if btn_on = '1' then --testing button on Nexys 4
                alarm <= '1';
                trigger <= "0001"; --4-bit word for Arduino
                arduino <= '0'; --to be sure, it is 0
            end if;

            if PIR = '1' then
                alarm <= '1';
                trigger <= "0010"; --4-bit word for Arduino
                arduino <= '0';  --to be sure, it is 0
                trigger_disp := (17,16,18,21); --PIR
            end if;

            if reed = '1' then
                alarm <= '1';
                trigger <= "0100"; --4-bit word for Arduino
                arduino <= '0'; --to be sure, it is 0
                trigger_disp := (18,14,14,13); --reed
            end if;

            if flame = '1' then
                alarm <= '1';
                trigger <= "1000"; --4-bit word for Arduino
                arduino <= '0'; --to be sure, it is 0
                trigger_disp := (15,16,18,14); --FIrE
            end if;

        else
            RGB_status <= "010";
            --Green diode, unlocked, cannot be triggered
        end if;

        if btn_off = '1' then --servis button, turn off alarm
            alarm <= '0';
            lock <= not lock;
            i <= 0;
            guessed := (21,21,21,21);
```

```
            trigger_disp := (21,21,21,21);
        end if;

        --matrix keypad
        case temp(16 downto 14) is
        when "000" =>
        row_line <= "0111"; --first row

        when "001" =>
        if col_line = "0111" then           -- d
        if key_press(13) = '0' then  --indication pressed key
        key_press(13) <= '1';
        guessed(i) := 13;
        if i < 3 then         --digit number in 7-seg display
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "1011" then    -- #, ENTER
        if key_press(15) = '0' then
        key_press(15) <= '1';

        if guessed = password then  --correct password
        if alarm = '1' then
        alarm <= '0';
        lock <= not lock;
        i <= 0;
        guessed := (21,21,21,21);       --empty array
        trigger_disp := (21,21,21,21);  --empty array
        else
        i <= 0;
        lock <= not lock;
        guessed := (21,21,21,21);
        trigger_disp := (21,21,21,21);
        end if;
        RGB_password <= "010";  -- Green, Correct password
        else                    --incorrect password
        i <= 0;
        guessed := (21,21,21,21);
        RGB_password <= "100";  -- Red, Incorrect password
        end if;
        end if;

        elsif col_line = "1101" then        -- 0
```

```
        if key_press(0) = '0' then
        key_press(0) <= '1';
        guessed(i) := 0;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "1110" then       -- *
        if key_press(14) = '0' then
        key_press(14) <= '1';
        guessed(i) := 19;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        else
        key_press(0) <= '0';
        key_press(15 downto 13) <= "000"; --no key pressed

        end if;

        when "010" =>
        row_line <= "1011"; --second row

        when "011" =>
        if col_line = "1110" then        -- 7
        if key_press(7) = '0' then
        key_press(7) <= '1';
        guessed(i) := 7;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "1101" then        -- 8
        if key_press(8) = '0' then
        key_press(8) <= '1';
        guessed(i) := 8;
```

```
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "1011" then        -- 9
        if key_press(9) = '0' then
        key_press(9) <= '1';
        guessed(i) := 9;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "0111" then       -- c
        if key_press(12) = '0' then
        key_press(12) <= '1';
        guessed(i) := 12;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        else
        key_press(9 downto 7) <= "000";
        key_press(12) <= '0';
        end if;

        when "100" =>
        row_line <= "1101"; --third row

        when "101" =>
        if col_line = "1110" then        -- 4
        if key_press(4) = '0' then
        key_press(4) <= '1';
        guessed(i) := 4;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
```

```
        end if;
        end if;

        elsif col_line = "1101" then      -- 5
        if key_press(5) = '0' then
        key_press(5) <= '1';
        guessed(i) := 5;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "1011" then       -- 6
        if key_press(6) = '0' then
        key_press(6) <= '1';
        guessed(i) := 6;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "0111" then        -- b
        if key_press(11) = '0' then
        key_press(11) <= '1';
        guessed(i) := 11;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        else
        key_press(6 downto 4) <= "000";
        key_press(11) <= '0';

        end if;

        when "110" =>
        row_line <= "1110"; --fourth row

        when "111" =>
```

```
        if col_line = "1110" then          -- 1
        if key_press(1) = '0' then
        key_press(1) <= '1';
        guessed(i) := 1;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "1101" then         -- 2
        if key_press(2) = '0' then
        key_press(2) <= '1';
        guessed(i) := 2;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "1011" then       -- 3
        if key_press(3) = '0' then
        key_press(3) <= '1';
        guessed(i) := 3;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        elsif col_line = "0111" then       -- A
        if key_press(10) = '0' then
        key_press(10) <= '1';
        guessed(i) := 10;
        if i < 3 then
        i <= i + 1;
        else
        i <= 0;
        end if;
        end if;

        else
        key_press(3 downto 1) <= "000";
```

```
        key_press(10) <= '0';

        end if;

        when others => led <= "0000000000000000";
        --for case of unexpected state

        end case;
        status <= alarm;
        arduino <= alarm;
    end if;

    --7-seg display
    if (clock = '1' and clock'event) then
    case (count_7s(19 downto 17)) is
    when "000" => digit <= "11110111"; --digits switching
    case (guessed(3)) is
    when 0 => segment <= nula;
    when 1 => segment <= jedna;
    when 2 => segment <= dva;
    when 3 => segment <= tri;
    when 4 => segment <= ctyri;
    when 5 => segment <= pet;
    when 6 => segment <= sest;
    when 7 => segment <= sedm;
    when 8 => segment <= osm;
    when 9 => segment <= devet;
    when 10 => segment <= A;
    when 11 => segment <= B;
    when 12 => segment <= C;
    when 13 => segment <= D;
    when 14 => segment <= E;
    when 15 => segment <= F;
    when 16 => segment <= ii;
    when 17 => segment <= P;
    when 18 => segment <= r;
    when 19 => segment <= star;
    when 20 => segment <= hash;
    when 21 => segment <= prazdny;
    when others => segment <= prazdny;
    end case;
    when "001" => digit <= "11111011";
    case (guessed(2)) is
    when 0 => segment <= nula;
    when 1 => segment <= jedna;
    when 2 => segment <= dva;
```

```
    when 3 => segment <= tri;
    when 4 => segment <= ctyri;
    when 5 => segment <= pet;
    when 6 => segment <= sest;
    when 7 => segment <= sedm;
    when 8 => segment <= osm;
    when 9 => segment <= devet;
    when 10 => segment <= A;
    when 11 => segment <= B;
    when 12 => segment <= C;
    when 13 => segment <= D;
    when 14 => segment <= E;
    when 15 => segment <= F;
    when 16 => segment <= ii;
    when 17 => segment <= P;
    when 18 => segment <= r;
    when 19 => segment <= star;
    when 20 => segment <= hash;
    when 21 => segment <= prazdny;
    when others => segment <= prazdny;
    end case;
    when "010" => digit <= "11111101";
    case (guessed(1)) is
    when 0 => segment <= nula;
    when 1 => segment <= jedna;
    when 2 => segment <= dva;
    when 3 => segment <= tri;
    when 4 => segment <= ctyri;
    when 5 => segment <= pet;
    when 6 => segment <= sest;
    when 7 => segment <= sedm;
    when 8 => segment <= osm;
    when 9 => segment <= devet;
    when 10 => segment <= A;
    when 11 => segment <= B;
    when 12 => segment <= C;
    when 13 => segment <= D;
    when 14 => segment <= E;
    when 15 => segment <= F;
    when 16 => segment <= ii;
    when 17 => segment <= P;
    when 18 => segment <= r;
    when 19 => segment <= star;
    when 20 => segment <= hash;
    when 21 => segment <= prazdny;
    when others => segment <= prazdny;
```

```
    end case;
    when "011" => digit <= "11111110";
    case (guessed(0)) is
    when 0 => segment <= nula;
    when 1 => segment <= jedna;
    when 2 => segment <= dva;
    when 3 => segment <= tri;
    when 4 => segment <= ctyri;
    when 5 => segment <= pet;
    when 6 => segment <= sest;
    when 7 => segment <= sedm;
    when 8 => segment <= osm;
    when 9 => segment <= devet;
    when 10 => segment <= A;
    when 11 => segment <= B;
    when 12 => segment <= C;
    when 13 => segment <= D;
    when 14 => segment <= E;
    when 15 => segment <= F;
    when 16 => segment <= ii;
    when 17 => segment <= P;
    when 18 => segment <= r;
    when 19 => segment <= star;
    when 20 => segment <= hash;
    when 21 => segment <= prazdny;
    when others => segment <= prazdny;
    end case;
    when "100" => digit <= "01111111";
    case (trigger_disp(3)) is
    when 0 => segment <= nula;
    when 1 => segment <= jedna;
    when 2 => segment <= dva;
    when 3 => segment <= tri;
    when 4 => segment <= ctyri;
    when 5 => segment <= pet;
    when 6 => segment <= sest;
    when 7 => segment <= sedm;
    when 8 => segment <= osm;
    when 9 => segment <= devet;
    when 10 => segment <= A;
    when 11 => segment <= B;
    when 12 => segment <= C;
    when 13 => segment <= D;
    when 14 => segment <= E;
    when 15 => segment <= F;
    when 16 => segment <= ii;
```

```
    when 17 => segment <= P;
    when 18 => segment <= r;
    when 19 => segment <= star;
    when 20 => segment <= hash;
    when 21 => segment <= prazdny;
    when others => segment <= prazdny;
    end case;
    when "101" => digit <= "10111111";
    case (trigger_disp(2)) is
    when 0 => segment <= nula;
    when 1 => segment <= jedna;
    when 2 => segment <= dva;
    when 3 => segment <= tri;
    when 4 => segment <= ctyri;
    when 5 => segment <= pet;
    when 6 => segment <= sest;
    when 7 => segment <= sedm;
    when 8 => segment <= osm;
    when 9 => segment <= devet;
    when 10 => segment <= A;
    when 11 => segment <= B;
    when 12 => segment <= C;
    when 13 => segment <= D;
    when 14 => segment <= E;
    when 15 => segment <= F;
    when 16 => segment <= ii;
    when 17 => segment <= P;
    when 18 => segment <= r;
    when 19 => segment <= star;
    when 20 => segment <= hash;
    when 21 => segment <= prazdny;
    when others => segment <= prazdny;
    end case;
    when "110" => digit <= "11011111";
    case (trigger_disp(1)) is
    when 0 => segment <= nula;
    when 1 => segment <= jedna;
    when 2 => segment <= dva;
    when 3 => segment <= tri;
    when 4 => segment <= ctyri;
    when 5 => segment <= pet;
    when 6 => segment <= sest;
    when 7 => segment <= sedm;
    when 8 => segment <= osm;
    when 9 => segment <= devet;
    when 10 => segment <= A;
```

```
    when 11 => segment <= B;
    when 12 => segment <= C;
    when 13 => segment <= D;
    when 14 => segment <= E;
    when 15 => segment <= F;
    when 16 => segment <= ii;
    when 17 => segment <= P;
    when 18 => segment <= r;
    when 19 => segment <= star;
    when 20 => segment <= hash;
    when 21 => segment <= prazdny;
    when others => segment <= prazdny;
    end case;
    when "111" => digit <= "11101111";
    case (trigger_disp(0)) is
    when 0 => segment <= nula;
    when 1 => segment <= jedna;
    when 2 => segment <= dva;
    when 3 => segment <= tri;
    when 4 => segment <= ctyri;
    when 5 => segment <= pet;
    when 6 => segment <= sest;
    when 7 => segment <= sedm;
    when 8 => segment <= osm;
    when 9 => segment <= devet;
    when 10 => segment <= A;
    when 11 => segment <= B;
    when 12 => segment <= C;
    when 13 => segment <= D;
    when 14 => segment <= E;
    when 15 => segment <= F;
    when 16 => segment <= ii;
    when 17 => segment <= P;
    when 18 => segment <= r;
    when 19 => segment <= star;
    when 20 => segment <= hash;
    when 21 => segment <= prazdny;
    when others => segment <= prazdny;
    end case;
    end case;
    end if;
end process;
end Behavioral;
```

Příloha B

Arduino kód

```
#include <SD.h>       //library for communication with SD card

int LED = 13;         //signalization LED
int LED3 = 7;         //LEDs displaying trigger array
int LED2 = 6;
int LED1 = 5;
int LED0 = 4;
int alarmPin = 22;    //pins for communication between boards
int triggerPin = 23;
const int CS = 53;    //chip select on pin 53

int alarmState = 0;   //variables for edges detection
int lastAlarmState = 0;

int trigger[4];       //array for communication with Nexys

String data = "test";    //string for printing to SD card

void setup() {
    SD.begin(CS);         //initialization of SD card
pinMode(LED, OUTPUT);     //declaration of I/O pins
pinMode(LED3, OUTPUT);
pinMode(LED2, OUTPUT);
pinMode(LED1, OUTPUT);
pinMode(LED0, OUTPUT);
pinMode(alarmPin, INPUT);
pinMode(triggerPin, INPUT);
}

void loop() {
alarmState = digitalRead(alarmPin);

if (alarmState != lastAlarmState) {
if (alarmState == HIGH) {
delay(1);
```

```
for (int i=0; i<4; i++) {                    //read form Nexys 4
trigger[i] = digitalRead(triggerPin);
delay(1);
}

//read data
if (trigger[0] == 1 & trigger[1] == 0 & trigger[2] == 0
& trigger[3] == 0) {
data = "Button";
}

if (trigger[0] == 0 & trigger[1] == 1 & trigger[2] == 0
& trigger[3] == 0) {
data = "PIR";
}

if (trigger[0] == 0 & trigger[1] == 0 & trigger[2] == 1
& trigger[3] == 0) {
data = "Reed";
}

if (trigger[0] == 0 & trigger[1] == 0 & trigger[2] == 0
& trigger[3] == 1) {
data = "Fire";
}

//open file on SD card
File file = SD.open("log.txt", FILE_WRITE);
if (file) {
file.print("Alarm␣triggered␣-␣");           //print to file
file.println(data);
file.close();
}

//signalization that it is writing on SD card
digitalWrite(LED,HIGH);
delay(500);
digitalWrite(LED,LOW);
}

else {
File file = SD.open("log.txt", FILE_WRITE);
if (file) {
file.println("Password␣correct␣-␣Alarm␣off");
file.close();
}
```

```
for (int i=0; i<(sizeof(trigger) / sizeof(int)); i++) {
trigger[i] = 0;
}

//signalization that it is writing on SD card
digitalWrite(LED,HIGH);
delay(1000);
digitalWrite(LED,LOW);
}
}
lastAlarmState = alarmState;

//displaying the state of trigger array
digitalWrite(LED3,trigger[3]);
digitalWrite(LED2,trigger[2]);
digitalWrite(LED1,trigger[1]);
digitalWrite(LED0,trigger[0]);
}
```

Příloha C

Obsah přiloženého CD

- VHDL Vivado projekt
- Arduino kód
- Datasheets
- Fotografie